Preprint
UCRL-JC-147446

Micro Benchmarking, Performance Assertions and Sensitivity Analysis: A Technique for Developing Adaptive Grid Applications

I. R. Corey, J. R. Johnson, J. S. Vetter

This article was submitted to
11th International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 24-26, 2002

February 25, 2002

Micro Benchmarking, Performance Assertions And Sensitivity Analysis:

A Technique For Developing Adaptive Grid Applications[*]

I.R. Corey, J.R. Johnson, J.S. Vetter
Computing Applications & Research Department
Lawrence Livermore National Laboratory

1) Abstract

This study presents a technique that can significantly improve the performance of a distributed application by allowing the application to locally adapt to architectural characteristics of distinct resources in a distributed system. Application performance is sensitive to *application parameter – system architecture* pairings. In a distributed or Grid enabled application, a single parameter configuration for the whole application will not always be optimal for every participating resource. In particular, some configurations can significantly degrade performance. Furthermore, the behavior of a system may change during the course of the run. The technique described here provides an automated mechanism for run-time adaptation of application parameters to the local system architecture. Using a simulation of a Monte Carlo physics code, we demonstrate that this technique can achieve speedups of 18% - 37% on individual resources in a distributed environment

[*]This work was performed under the auspices of the U. S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

2) Introduction

The interaction between the parameterization of an application and the architectural characteristics of the system on which the application is running is of fundamental importance to the overall performance of the application. For a distributed application, performance inequities between different resources due to favorable and unfavorable *application parameter – system architecture* pairings can significantly affect overall performance.

There is currently no general theoretical model for predicting performance based on parameter – architecture pairings. However there has been success with using empirical methods, ([1], [5], [8], [11]). These empirical systems perform extensive testing of a range of parameter values when they are installed on distinct architectures. In a heterogeneous, distributed environment, performing the empirical test at installation time is neither practical nor is it always possible. A single parameter configuration for the whole application will not always be optimal for every resource participating in a distributed computation. Resource characteristics may not be known at scheduling time and resources may come and go throughout the computation. In addition, executables may be staged rather than existing as highly tuned software already installed on a resource.

By using a combination of micro benchmarking, performance assertions and sensitivity analysis, the technique presented here allows a locally scheduled application component to adapt at run-time to the resource to which it has been assigned and then continue monitoring its performance and adapt as necessary. This technique is an extension of earlier empirical studies in that it operates on a large-scale application and is applied at run-time rather than install-time.

The technique works as follows:
(see Figure 1)

The application code is instrumented with performance assertions at positions in the code where system and problem parameters can be dynamically varied, (e.g. between time-steps, or modes such as grid generation and time dependent simultation).

When an instance of the application is deployed on a local resource in a distributed environment, it first runs a micro benchmark varying problem parameters one at time and finds the parameters that have the greatest affect on performance. This micro benchmark can either be performed within the application itself or by a separate benchmarking tool included as a library to the application.

From the aggregate benchmark data the best parameter configuration for the local resource is determined and the application is modified to use this configuration. In addition, performance assertion bounds and handlers are set using the micro benchmark results.

After the micro benchmark phase is complete and the configured application is running if the code fails to meet a performance assertion based on the original micro benchmark, then control is transferred to a function registered with the performance assertion that decides whether or not to modify a parameter value. Should the function determine that it is necessary to vary a parameter value, then additional benchmarking is performed beginning with the parameters to which performance is most sensitive. A new configuration is constructed and the application is modified to reflect the changes in the system behavior.

3) Related Work

An initial attempt at a formal framework for parameterized architecture adaptive algorithms was described by Ueberhuber and Krommer, ([9]). More recently, tangible empirical results on homogeneous, static architectures have been presented in ([1], [5], [8], [11]). NetSolve, ([2]), is a network tool designed to map a problem to the best available resource in a distributed environment based on network performance. NWS, ([12]), and Globus ([4], http://www.globus.org) provide information on the status of networks and the availability of servers. The GrADS project (http://nhse2.cs.rice.edu/grads/) is addressing issues of application performance and performance contracts on computational grids. Recently the GrADS project has presented a framework for adaptive Grid programs, ([7]). Code instrumentation and dynamic steering is explored in ([6], [10]).

4) Monte Carlo Simulation

The simulation used here is designed to emulate a large-scale Monte Carlo physics code under development at LLNL. Both the actual code and the simulation are structured in a *communicate, work, reduce* cycle. This iterative structure simplifies the adaptive process by allowing the adaptation to take place between iterations.

For this experiment only MPI parameters were investigated. Both the simulation and the Monte Carlo code have been constructed so that it is easy to dynamically change MPI parameters (e.g. type of send, buffer size, etc...).

5) Micro Benchmark Description

For the results described here, the message size was held constant at 1MB. The parameters varied were: *type of send*, *message factor* and *send buffer*.

Type of send was varied between *irsend*, *ssend*, *issend*. *Message factor*, is the number of equally sized units by which the message is divided. For example, with a 1MB message, a *message factor* of 2 sends 2 512K messages. A *message factor* of 4 sends 4 256K messages, etc.. The results presented here use *message factors* of 1, 2, 4 and 8, (for runs on LLNL's GPS cluster message factors of only 1,2,and 4 were used). *Send buffer* is the number of sends held in the buffer before waiting. The results presented here use *send buffers* of 0, 4, 7, and 10. Holding all other parameters constant and just varying these three one at a time generated a total of 54 tests (4*4*3) per run on ASCI Blue and 36 tests (4*3*3) per run on GPS.

These tests were run on two distinct architectures: 3 nodes/2 processors per node of IBM's ASCI Blue Pacific and 4 processors on a single node on LLNL's GPS Cluster (Compaq ES45 with 4 1GHz EV6.8 processors). For each test we ran the code for 100 iterations and measured wall-clock time, MPI-time and MB/s.

6) Results

Results described here are from multiple runs using the simulation code.

The results show distinctively that the instance of the simulation running on ASCI Blue Pacific was most sensitive to send-type with the slowest case of *irsend(mf1)* offering a speedup of 18% over the fastest *issend* with all other parameters held constant. (Table 1 & Table 2) The slowest *ssend* outperformed *issend* on the IBM averaging by nearly 20%. The split between *irsend* with message factor 1 and the other *irsend* data is most likely more an artifact some coding decisions in the simulation rather an indicator of how the Monte Carlo code would interact with the system configuration. Although, even if further tests show it to be an accurate measurement, it is interesting in the sense that it is a clearly distinguished behavior and it varies from behavior seen in the GPS micro benchmarks.

Type of Send	Max	Min
irsend (mf=1)	142.295	140.66
irsend (mf > 1)	272.17	269.83
ssend	181.363	179.302
issend	242.815	207.753

Table 1. *ASCI Blue wall-clock time per type of send*

Type of Send	Max MB/s	Min MB/s
irsend (mf=1)	37.798	37.321
irsend (mf > 1)	20.059	19.866
ssend	31.573	31.185
issend	26.211	22.112

Table 2. *ASCI Blue throughput per type of send.*

The micro benchmarks on GPS were not as separable as those on ASCI Blue. (Table 3) This is probably due to the fact that dedicated nodes were used for runs on ASCI Blue but on GPS the nodes were shared so load may have added some noise into the data. However, good results are still observable when averaging values across test runs. For the same code, GPS showed a similar profile for *send-type* but *message factor* had a more significant effect on performance with an average of 37% speedup by increasing *message factor* from 1 to 2 and an average speedup of 13% going from message factor 2 to message factor 4.

Message Factor	Ave MB/s
1	19.72
2	16.42
4	14.47

Table 3. *GPS Throughput per type of send.*

7) Conclusions

This work shows that for a given application, different architectures exhibit sensitivity to different parameters in a program's configuration. By changing the appropriate parameters for the given architecture, performance can be significantly improved. In the simple simulation studied here, speedup of from 18 % to 37% on a single resource is achieved.

While a good configuration for a particular architecture can often be discovered analytically, the complexity of building a rules system to handle every possible scenario can be unwieldy. Sometimes results are unexpected but not unreasonable (such as buffered sends performing better than unbuffered sends). This complexity and the fact that in a heterogeneous distributed computing environment, resource characteristics may not be known even at run-time lead to the conclusion that an empirical approach is the best for dynamic application configuration. The simulation described here shows that running application-specific micro benchmarks on the actual system offers a good mechanism for determining the effect of parameter-architecture pairings thus allowing the application to run using a good configuration on the local system.

For complex codes, a micro benchmark such as the one described here is a good tool for discovering the behavior of a system. For codes that have clear delineation of behavior (e.g. a computation phase followed by a communication phase), the code itself may be used to perform the benchmark and then adapt itself at runtime.

One of the challenges in evaluating micro benchmarks is choosing a metric to evaluate the system. Wall clock time is not always the best metric since many different things can affect it (e.g. load). The same is true about using throughput as a measure. This difficulty can be addressed by including data from system utilities such as hardware performance counters in the benchmarking phase. This benchmark data can be used to construct performance assertions that are inserted between time-steps to determine whether the behavior of the system is consistent with the original micro benchmark. If not, the performance assertion will execute a handler function to rerun the micro benchmark and re-configure the application if necessary.

8) Future Work

The full Monte Carlo physics code is currently being instrumented using this technique and results should be available in the next few weeks. This instrumentation includes performance assertions to monitor the application performance and re-configure if necessary. Test runs on the Monte Carlo code will be performed on dedicated nodes of GPS to alleviate any anomalies introduced due to load on the machines.

The results presented here were based on a handful of MPI parameters. Both the simulation and the Monte Carlo code can vary many more MPI parameters than those described here. In addition there are non-MPI parameters that may also significantly affect performance (e.g. threads, problem size, array strides and blocking etc...). Choosing the best parameters to use in the micro benchmark is also an area for further exploration.

Over the next few months this technique will be implemented in a fully distributed testbed to explore how the local adaptation contributes to overall performance improvement.

9) Bibliography

[1] J. Bilmes, K. Asanovic, C-W. Chin, J. Demmel, "Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology.", *International Conference on Supercomputing*, 1997.

[2] H. Casanova, J. Dongarra. "NetSolve; A Network Server for Solving Computational Science Problems.", *International Journal for Supercomputer Applications and High Performance Computing*, vol. 11, no. 3, 1997.

[3] I. Foster, C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco, 1998.

[4] I. Foster, C. Kesselman, Globus: "A Metacomputing Instrastructure Toolkit." *International Journal of Supercomputer Applciations*, vol. 11, no. 2, 1997.

[5] M. Frigo and S. G. Johnson, "FFTW: An Adaptive Software Architecture For The FFT." *ICASSP*, vol. 3, 1998.

[6] J. K. Hollingsworkth, P. Keleher, "Prediction and Adaptation in Active Harmony", *Cluster Computing*, vol. 2, 1999.

[7] K. Kennedy, M. Mazina, et. al., "Toward a Framework for Preparing and Executing Adaptive Grid Programs", *International Parallel and Distributed Processing Symposium*. 2002. (to appear)

[8] D. Mirkovic, S. L. Johnsson, "Automatic Performance Tuning in the UHFFT Library", *International Conference on Parallel Computing*, 2001.

[9] A. R. Krommer, C.W. Ueberhuber, "Architecture Adaptive Algorithms", *Parallel Computing*, vol. 19, 1993.

[10] J. S. Vetter, D. A. Reed, "Real-time Performance Monitoring, Adaptive Control and Interactive Steering of Computational Grids", *The International Journal of High Performance Computing Applications*, vol. 14, no.4, 2000.

[11] R. C. Whaley, A. Petitet, J. J. Dongarra, "Automated Emprical Optimizations of Software and the ATLAS Project." *Parallel Computing*, vol. 27, no1-2, 2001

[12] R. Wolski, N. Spring, J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing." *Journal of Future Generation Computing Systems*, vol.15, no. 5-6, 1999.

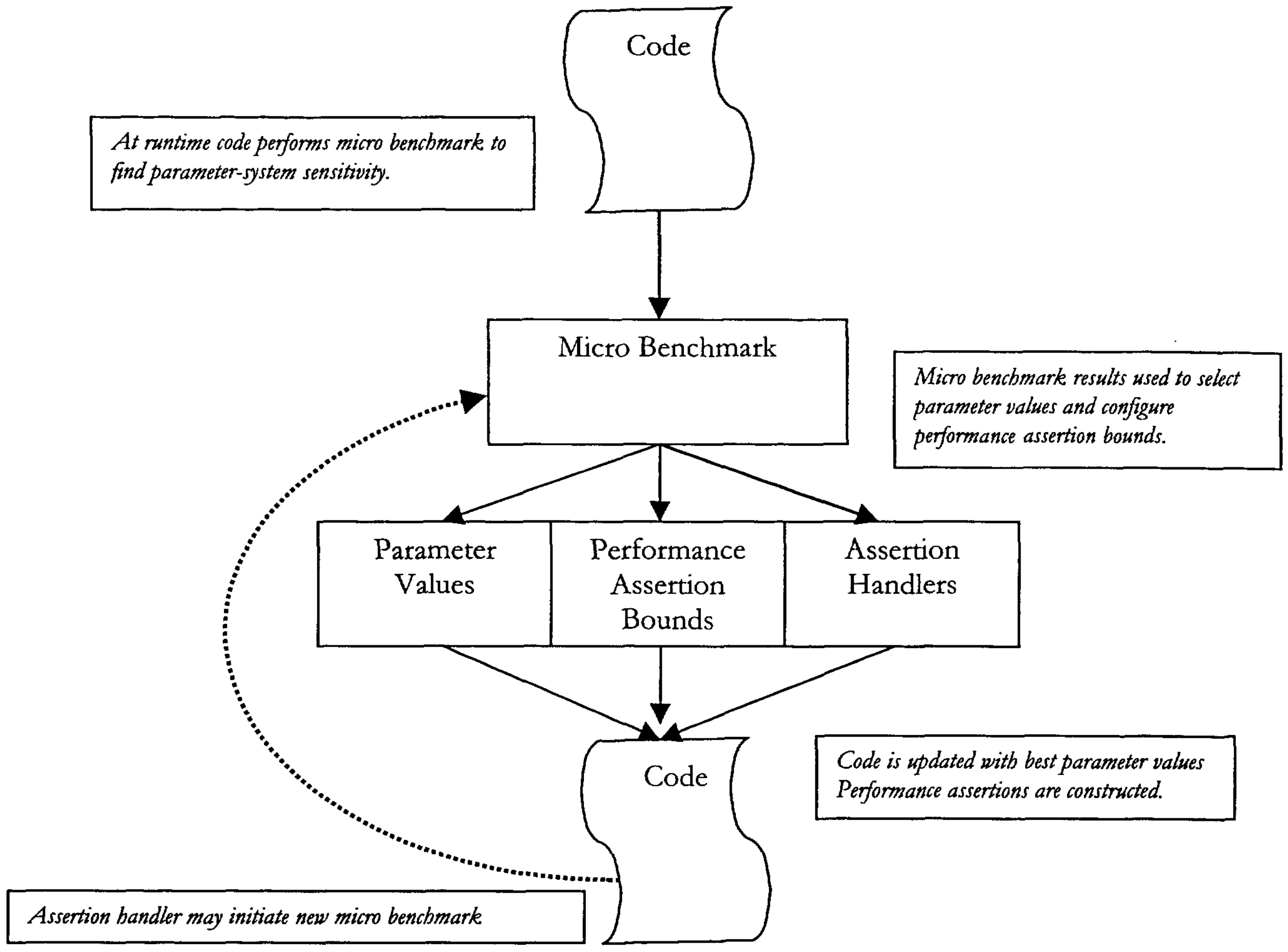
Code
At runtime code performs micro benchmark to find parameter-system sensitivity.
Micro Benchmark
Micro benchmark results used to select parameter values and configure performance assertion bounds.
Parameter Values
Performance Assertion Bounds
Assertion Handlers
Code
Code is updated with best parameter values Performance assertions are constructed.
Assertion handler may initiate new micro benchmark

Figure 1.